Setup

- define PV and BESS sizes
- create auxiliary dictionaries (seasons, months, day-types and days distributions)
- create time dictionary (time discretisation)
- import battery' specifications
- define BESS flag (if size != 0)
- create technologies dictionary (sizes and specifications)
- import production profiles; broadcast production profiles over day-types
- import electric load profiles
- call *configuration_evaluation* to get yearly values of shared energy, injections, withdrawals

configuration_evaluation

1. Inputs
   - store time discretisation's variables
   - store auxiliary variables
   - store technologies variables
   - define BESS flag if BESS is in technologies
   - store production (**pv_production**)/load (**ue_demand**) profiles arrays
2. Evaluation
   - initialise arrays for monthly energy values of shared energy, injections, withdrawals
   - if BESS flag is False:
     - injections = production, **withdrawals** = **ue_demand**,
     - shared = *np.minimum*(**injections**, **withdrawals**)
     - evaluate monthly values
   - else:
     - for each typical day (month and day-type):
       - call *power_flows_optimisation* to get daily profiles of shared power, injections, withdrawals
       - if optimisation fails:
         - activate failed optimisation flag; continue
       - else:
         - evaluate energy values and store in monthly values
     - if any failed optimisation flag is active:
       - for each typical day where the flag is activated:
         - if the flag is activated for both day-types in the month:
           - store energy values as nans
         - else:
           - fix values using adjacent typical day
       - if any nan value in monthly energy arrays:
         - fill nans interpolating between months
3. Return
   - return yearly values of shared energy, injections, withdrawals

power_flows_optimisation

1. Inputs
   - store time discretisation's variables
   - store technologies variables
   - define BESS flag if BESS is in technologies
   - store production (**pv_production**)/load (**ue_demand**) profiles arrays
2. Evaluation
   - if BESS flag is False:
     - activate easy solution flag
   - evaluate **excess** = *np.maximum*(**pv_production** – **ue_demand**, 0)
     - if all **excess** is 0:
       - activate easy solution flag:
   - if easy solution flag is activated:
     - **injections** = **production; withdrawals** = **ue_demand**,
     - **shared_power** = *np.minimum*(**injections**, **withdrawals**)
     - return **shared_power**, **injections**, **withdrawals** and status of optimisation ('unneeded')
   - build the optimisation problem:
   - try to solve the optimisation problem
   - except: return nans and optimisation status ('failed')
   - else:
     - evaluate injections = **pv_production** + **battery_discharge** – **battery_charge**
     - evaluate withdrawals = **ue_demand**
     - return results and optimisation status ('optimal')